**Sequential Matrix Multiplication**

**Introduction:**

Matrix multiplication is the most-studied algorithm in high performance computing. In this report, we will write C language code to compute **C = C + A \* B**, where **A**, **B**, **C** are NXN matrices which holds double floating point random numbers and N can be 10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 200, 300, 400, 500, 600, 700, 800, 900, 1000. Now matrix multiplication can be computed for six combinations of ijk which are ijk, ikj, jik, jki, kij, kji. After writing six functions to compute matrix multiplication, we can calculate and compare execution time, floating point operation per second (flop/s). We can also use PAPI library to analyze the reason behind such differences.

**Method:**

At the very beginning, **A**, **B**, **C** matrices are dynamically memory allocated and randomly initialized for various values of N. Instead of initializing array as 2D, they are initialized as 1D or linearly to get better performance. After that matrix multiplication and addition is performed for various ijk combinations. To calculate execution time of each combination, default time function of C language is used. But to get better precision, PAPI library’s time function is utilized. It is worth to point that only the execution time to compute the matrix multiplication is calculated, not memory allocation time or randomly matrix initialization time. While calculating the execution time, the computation function for a combination is called NUM (here NUM=5) times to get the average execution time for that combination of any specific N. The below chart shows execution time of different combinations of matrix multiplication for various N. From the chart, it is evident that IJK, JIK, KIJ have similar execution and which are almost always more IKJ, JKI, KJI combinations which have similar execution time.

From the execution time, we can calculate using the formula provided: flop/s = 2N3/time. We can divide it by 109 to get Gflop/s. The below chart shows Gflop/s of different combinations of matrix multiplication for various N. JIK combination for N=100 has the maximum **2.945508 Gflop/s**.

To investigate more, we used PAPI library to analyze the reason behind this performance fluctuation. We used few events (PAPI\_L1\_TCM, PAPI\_L2\_TCM, PAPI\_L1\_TCH, PAPI\_L1\_TCA) available from papi\_avail command found at BigRedII to measure the L1, L2 cache hit/miss condition. We calculated miss rate by this formula:

*L1 cache miss rate = PAPI\_L1\_TCM/PAPI\_L1\_TCA = L1 Total Cache Miss/L1 Total Cache Access*

As we realize from the previous findings that IJK, JIK, KIJ combinations have similar execution and flop/s, so we will create two separate charts for two groups of combination to better visualize them. The below chart shows miss rate of IJK, JIK, KIJ combinations. It is important to note that from N=300, the miss rate jumps substantially.

The below charts shows miss rate of IKJ, JKI, KJI combinations. It is important to note that, from N=10 to N=20, the miss rate drops significantly.

All the computations are don’t at IUPUI’s BigRed2 super computer whose clock rate is known as 2.5GHz, so it’s peak performance = 4\*CPU clock rate = 4\*2.5 = 10. Our best program is JIK combination for N=100 which has the maximum **2.945508 Gflop/s**. So it utilizes around 30% of the peak performance.

**Conclusion:**

From PAPI analysis results, we can find that after N=200, (IJK, JIK, KIJ) combination has significant cache miss rate, but for (IKJ, JKI, KJI) combination has consistent cache miss rate from N=20. This is also evident from Gflop/s chart which shows that before N=200, (IJK, JIK, KIJ) combination performed higher floating point operation per second, but after that the flop/s decreased due to higher cache miss rate which eventually increased higher execution time. On the other hand, (IKJ, JKI, KJI) combination has consistent Gflop/s due to consistent cache miss rate.